TechTrends
https://doi.org/10.1007/s11528-022-00782-1

ORIGINAL PAPER

Real Coding and Real Games: Design and Development of a Middle School Curriculum Using Unity 3D

Mete Akcaoglu[1] · **Selcuk Dogan**[1] · **Charles B. Hodges**[1]

Accepted: 10 September 2022

Abstract
In this paper, we describe the design, development, and implementation of a curriculum based on teaching computer science using an industry-standard game-design software: Unity 3D. We discuss the theoretical underpinnings of our instructional design process and steps we have taken to introduce complexity and maintain student motivation. We discuss the challenges of this implementation and possible solutions and detail the additional steps related to teacher professional development, which is a key element for success of new and innovative curricular implementations such as ours.

Keywords Game-design · Unity3D · Computer science education · C#

Computer technology has become an indispensable part of our lives. In recent years, our interaction with computers has become more integrated, requiring us to talk the language of computers (Sullivan & Denner, 2017) to command our smart home devices by creating automations, or creating task flows in our computers to do certain tasks more effectively. Although such tasks may not directly require coding, they require understanding the computational principles to become computationally literate. Outpacing the home integration, computational literacy is already becoming a requirement for many jobs, and the job market for people with these skills will only grow bigger.

Despite the national push for computer science (CS) education, however, the progress is still slow: few teachers have the capacity to teach CS, and there is a disproportionate attendance from underrepresented groups (Sullivan & Denner, 2017). The progress is even slower in contexts with sizable low-socioeconomic status (SES) and disadvantaged student populations. Despite the fact that CS jobs can be provide paths for economic mobility, the number of students pursing CS education and careers are moving slowly.

✉ Mete Akcaoglu
makcaoglu@georgiasouthern.edu

Selcuk Dogan
sdogan@georgiasouthern.edu

Charles B. Hodges
chodges@georgiasouthern.edu

[1] College of Education, Georgia Southern University, Statesboro, GA 30458, USA

To address the lack of opportunities in CSEd, we must find ways to provide practicing teachers with high-quality professional development (PD) and support to teach CS in K-12 schools. Although CS PD opportunities are sporadically available through local Regional Education Service Agencies or nation-wide (e.g., online workshops), the majority are not sustainable, insufficient in duration, do not focus clearly on teacher-specific CS knowledge, or offer sustained support (Menekse, 2015). PD should be high-quality, structured, and engaging so that it leads to changes in teacher knowledge, practices, and eventually improvements in student outcomes (Goode et al., 2014). But PD alone will not solve the problem; we also need to work with the school systems to form functioning Research and Practitioner Partnerships (RPPs) to create research-informed practices that are tailored to meet local and national needs. In our local context, despite a push for partnership between university (researchers) and practitioners (teachers, school leaders, and district officials), these parties have been historically disconnected in their efforts.

In addition to effective PD opportunities for teachers and forming an RPP, preparing today's students for success in CS education and future employment in the CS fields requires designing learning environments that are relevant and attractive to all students. To provide a more accessible CS curriculum that appeals to a wider range of students, especially in low-SES, low-motivation contexts, methods that have benefits over traditional CS instruction need to be developed. Among the programs that have gained momentum in CSEd is the use of game-design activities not only to teach CS but also to

introduce students to an engaging and meaningful context for design and higher-order thinking skills (Akcaoglu, 2014; Denner et al., 2012; Kafai & Burke, 2016). NSF-funded efforts, like Repenning's Scalable Game Design (Basawapatna et al., 2010), our previous work (e.g., Akcaoglu & Koehler, 2014), and state-wide efforts target teaching CS through game-design courses have demonstrated how educators can develop robust and engaging game-design based CS learning. Almost all the previous game-design CS efforts, however, incorporated software that were visually attractive but limited in their real-world usage or immediate relevance.

In this paper, we detail the design and development process of our game-design-based computer science middle school curriculum to increase their motivation and learning of CS, using an industry-standard game design software (Unity 3D) that has real-world relevance.

Why Game Design?

Digital game design has been an attractive and appropriate context to engage young students in higher-order thinking (e.g., Akcaoglu, 2014), explore STEM content (e.g., Baytak & Land, 2011), learn basic knowledge of and develop interest toward computer programming (Comber et al., 2019), and form positive attitudes toward CS (Denner et al., 2019). Embedding computer programming in meaningful media-rich project-based contexts has promise for increased equity: i.e., it helps decrease the gender gap in CS (Guzdial, 2015). Game-design activities are meaningful media-rich contexts and can, therefore, provide benefits in terms of broadening participation by encouraging participation of underrepresented students in CS (Denner et al., 2012; Werner et al., 2020). When incorporated into regular curricula, game-design courses help broaden participation by attracting students from different backgrounds into CS (Repenning et al., 2015).

Research on digital game design has unearthed its potential for teaching and learning, highlighting its suitability to teach CS. There are, however, a few key elements that have been lacking in game design instruction and research. First, most studies did not report key research elements and were not rigorously designed (Denner et al., 2019), which hinders knowledge-building in the field. Second, being offered as either after-school or summer camp activities (Repenning et al., 2015), most game-design courses reach a limited set of highly motivated students (Werner et al., 2020). Finally, in such informal settings, teachers usually are in support roles and not fully involved in design and teaching. Therefore, many do not incorporate game-design as a classroom activity after the research concludes. In Project GAME, we alleviated these shortcomings by (a) designing rigorous research around the development and learning outcomes, (b) embedding teaching into the formal school curriculum and offering it to a diverse body of students, and (c) giving teachers lead designer and teaching roles.

Why Unity over Block-based Software?

Instead of block-based software, in this project we used a professional game-design engine: Unity 3D. Block-based programming tools have been popular platforms to introduce students to CS (Grover & Basu, 2017) and digital game-design (Earp, 2015). Despite their popularity and wide use in game-design contexts, however, researchers and practitioners have also pointed to some shortcomings of block-based programming and similar "opaque" (i.e., the inner mechanisms are hidden from users) approaches to CS (Grover & Basu, 2017; Repenning, 2017; Meerbaum-Salant et al., 2011). One issue with a visually attractive, but opaque, CS approach is that while learners might benefit from a quick learning curve, they eventually come to a halt when they reach complexity (Repenning, 2017). For example, students creating games in Scratch were found to infrequently use advanced concepts like variables, loops, and Boolean operations (Grover & Basu, 2017). In addition, it was found that without appropriate pedagogical approaches (i.e., guided-discovery learning), due to the opaqueness in their approach, block-based tools can lead to misconceptions (Meerbaum-Salant et al., 2011). Finally, although block-based coding tools excel in providing syntactic support and, thus, ease entry to coding (Grover & Basu, 2017), they lack support in other essentials: semantics and pragmatics (Repenning, 2017). Just having tools for only syntax is akin to spell-checking features in word-processing software: they can make you a more accurate writer, but do not automatically help you to produce best-selling novels (Repenning, 2017). Therefore, we needed a game-design software and curriculum addressing these issues stated above.

Unity 3D is an industry-standard (Deals, 2016) cross-platform (over 25 platforms) game design engine built on C# programming language (Comber et al., 2019; Dickson, 2015; Unity, n.d.). It is popular: it has been installed on over three billion devices worldwide. Unity allows designers to develop games in multiple genres ranging from simple to complex, 2D to 3D, AR to VR. Unity employs a transparent approach to coding: code is in text form and its outcome is immediately shown in the game output. It has a low threshold and high ceiling (Repenning et al., 2010); the software allows the creation of beginner-level games as well as full-fledged games to be shared and played in many platforms the students have access to (including mobile). It has real-world and industry acceptability and relevance. Unity can scaffold flow (can be adjusted for challenge), enable transfer (transparent CS tasks), and can be designed to support equity (through game-design and other curriculum design and activities), which are important elements of CS tools (Repenning et al., 2010). Notably, Unity is free to educational institutions and students, and does not require special hardware: it can run on most basic computers.

Underpinnings of Project GAME Curriculum

Our curriculum was informed by an interrelated network of theories in learning and instructional design, curricular standards, student needs, and the desired learning outcomes the school districts value. Both our previous Game Design and Learning (GDL) curriculum and courses (e.g., Akcaoglu, 2014, 2016; Akcaoglu & Green, 2019; Akcaoglu & Kale, 2016; Akcaoglu et al., 2016; Akcaoglu & Koehler, 2014) that we taught to hundreds of K-12 students and preservice teachers in various formal and informal school settings to teach higher-order thinking and game-design skills, as well as Repenning's similar work with the Scalable Game Design (SGD) project (e.g., Repenning et al., 2015) will provide theoretical and practical design guidance. Below, we discuss how the specific applications from this previous work inform our current curriculum design and curriculum development model.

A common design element in both GDL and SGD is that the curriculum introduces the students to simple game-design tasks initially and incrementally increases the difficulty and breadth of content and skills covered (e.g., Akcaoglu, 2016). For example, in the initial lessons, students design a very simple game where they reach one goal and the game is completed, while, in the later stages, they develop games with more complex rules and goals requiring more advanced CS and game-design knowledge. Through such a "project-first" (Repenning et al., 2015) approach, students are rewarded with a tangible outcome at the end of each instructional experience. This approach provides students with an initial sense of accomplishment and gives them enough challenges and rewards for their progress instantly (i.e., Repenning et al.'s (2015) Zones of Proximal Flow framework). This approach also allows us to moderate or reduce the effects of the cognitive load (Mayer & Wittrock, 2006) or expertise reversal effect (Sweller, 2020) the software would otherwise pose. By keeping the initial tasks simpler, we ensure software is mastered before moving to cognitively-taxing tasks. Similarly, with hard skills mastered, in the final stages of the curriculum, time is devoted to more soft skill activities such as "free design" which allow both flexibility in teachers' curriculum implementation and a creativity to students who can independently choose the upper limits of complexity and develop a game that they personally value (Akcaoglu, 2016). To ensure CS knowledge is mastered, we provided appropriate scaffolding (i.e., guided discovery) by explicitly teaching concepts when needed. Such approaches have been effective teaching approaches in CS including game-design settings (Akcaoglu, 2014; Kirschner & Neelen, 2020).

In this project, we served low-SES and underserved students with low motivation and self-efficacy. Therefore, we also used a well-established instructional design approach: the Attention, Relevance, Confidence, and Satisfaction (ARCS) model (Li & Keller, 2018) to address the challenges in motivating students to learn. The ARCS model has been used successfully to design and develop learning experiences that increase learner motivation in game design, computer science (e.g. Tlili et al., 2017), and STEM disciplines for underrepresented groups (e.g. Bosman et al., 2017). In the design process, the team worked to determine ways to establish relevance for computer science and game-design to gain students' attention. For example, we established relevance by inviting individuals from the game-design industry (through our partnership with Unity) to present their work and talk about game-design and computer science careers. Students build their confidence by appropriately pacing the curriculum (i.e., ZPF). Student motivation and satisfaction will be maintained by helping students feel a sense of reward (i.e., "project-first"). Through this design, we hoped to improve students' motivation for CS, especially their utility value, self-efficacy, and interest. Finally, a central design element for our instructional activities was Campe et al.'s (2019) NSF-supported work: pair programming. As a CSEd approach, it has been beneficial for students' learning programming (Esquenazi, 2020; Hartl et al., 2015), especially for underrepresented populations (e.g., girls) or low-motivation students with low access to quality teaching as in our schools. Through pair programming, Denner et al. (2019) found that students benefited both cognitively (i.e., increased knowledge) and motivationally (i.e., increased self-efficacy). As such, based on the findings of existing research on pair-programming and the recent toolkit, our instructional activities were built on pair-programming.

The Curriculum Development Process

We followed a curriculum development model (Fig. 1), which includes input and design guidance to create draft unit plans, followed by curriculum revision through feedback from teachers and advisory board members. The curriculum was modular (along a continuum) in that it is separated by individual unit plans.

Our process included a co-development approach aligned with our effort to form a research practitioner partnership (RPP). We embraced Penuel et al.'s (2007) principles to structure our co-development process: "a highly facilitated, team-based process in which teachers, researchers, and a content expert work together in defined roles to design an educational innovation, realize the design in one or more prototypes, and evaluate each prototype's significance for addressing a concrete educational need" (p. 53). Through online and in-person means, teachers provided their diverse and practical expertise to co-develop and shape the curriculum and assist with revisions through their self-reflections.

During the Content Mapping Phase in online professional learning communities (PLC), researchers and the teachers first reviewed and created an overall map of CS concepts, game design skills, and drew the alignments to state and national CS standards. With the guidance of the CS content expert,

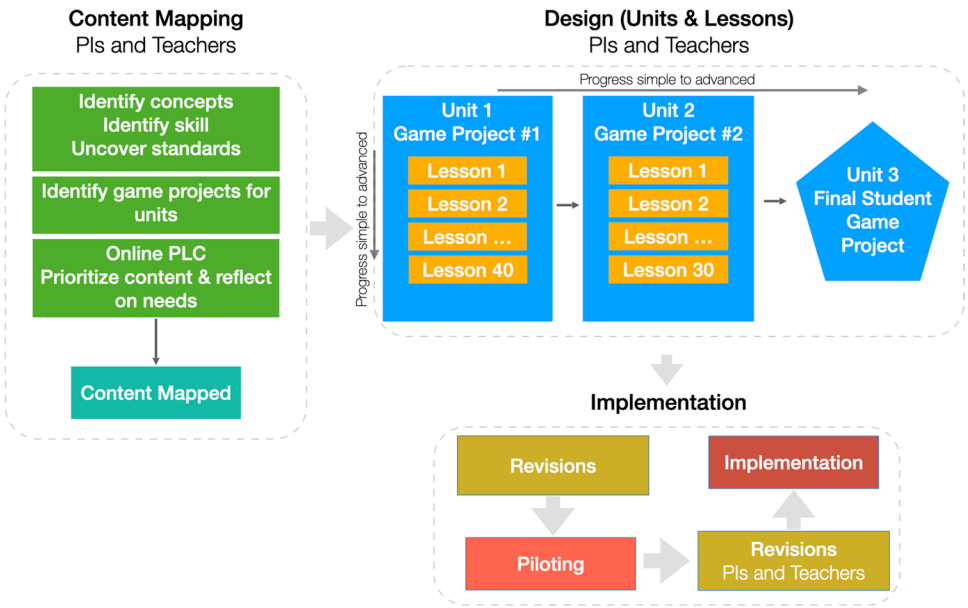

Fig. 1 Curriculum development process

they developed content outlines (principles and processes that should serve as the focal point of our curriculum), student learning outcomes, and learning objectives as well as decided the sequence of all in terms of how they would fit into game projects to progress from simple to advanced. Unity already had some units available that worked as a starting point in terms of CS and game-design skills and content to be covered, as well as some sample game projects. They used these as a baseline to identify possible game projects and the breakdown of game-design and CS concepts and topics to be covered in our curriculum map. Second, based on discussions and reflection from our teachers in the online discussion boards, they prioritized and adjusted the content based on the needs of the teachers and the target students (e.g., extended time devoted to a topic). Clarifying content priorities included categorizing the content according to the essential and big ideas as well as core tasks students were expected to complete.

In the Design Phase, our primary goal was to create draft units and learning activities. The researchers created a lesson plan template (sample below) and the initial draft lesson plans using design principles (ARCS, pair-programming). Each draft unit plan included learning objectives in various levels of cognitive domain and learning activities in which students were engaged to learn about CS concepts and game-design skills. For example, a lesson covered the conditionals (i.e., if-then statements) as a CS concept. After explaining this concept and making connections by giving real-life examples (e.g., if the light is red, then stop the car), the concept of conditionals was introduced in the game-design context. The assignment required students to limit the player from going out of bounds of the game space. To this end, students worked to write the code so that "if a bouncing ball touches the boundaries of the game space, then the ball will deflect back into the game space." Through a working example and explicit explanation and real-life connections, the students worked in pairs to integrate this component into their games. Learning assessments were in the form of students' authentic work (i.e., games or planning documents) graded using rubrics created by teachers during co-development.

Teachers also created and shared lessons within each unit for review. In online PLC, the teachers independently and as a group reviewed and provided feedback until all lessons were finalized. A design checklist and a structured protocol were used to scrutinize each element in the lessons. Our advisory board members (CS content experts) provided their input on the design and structure of the curriculum and its specific components and evaluated them in terms of their capacity to offer learning opportunities for CS during and toward the end of the CD process.

During the Implementation Phase, our teachers piloted the curriculum for one semester. The teachers implemented individual lessons in each unit and moved along the CS topics from simple to more complex. We revised the lessons based on teacher and student feedback during and after the pilot and implemented a revised version of it the following semester. A map of our curriculum along with all the lesson plans (and the accompanying videos) can be found on the project website: https://www.projectgame.org/.

Lesson Plan Sample

The following template shows what the teachers and students did in one lesson (Table 1). The lesson used the ARCS model and pair-programming elements, focusing on motivational aspects of instructional design for the high-need students, as well as covering essential CS and game-design skills. All the lesson plans and materials, as well as a curriculum map for the project can be accessed at https://projectgame.org.

CS Concept: if-statements in Unity 3D, game-design=boundaries

Table 1 Sample lesson plan and flow

Activity	Justification and Explanation
In the previous lesson, the teacher asked students the following guiding question: "How can we make sure the character does not go out of bounds in our game?" The teacher first seeks answers to the question from the day before. Then, they show an example of an *if-statement* (the character stops at the edge of the screen) and ask what's happening on the screen. 10 minutes	• The guiding question from the lesson before and the new behavior of the game character (stopping at the boundary) serve as a novelty for the students. With a unique ability added to their game, students' attention is attracted. • The teacher, then, goes into explaining the role of if-statements in coding and gives examples from real-life. A good example would be to "code" a student to stop at the classroom door. The teacher instructs a student to "stop if you touch the door," and the class watches one student perform the command. The teacher then elicits another command for the student to complete. This activity acts as the relevance. Students not only see the use of if-statements in coding, but they also experience how it applies to real-life settings.
Following the introduction, the teacher gives instructions for the activity that will take the rest of the lesson: - Students are to implement borders into their games by using if-then statements: if the bot touches the boundaries, it stops. 30 minutes	• Students work in pairs —set at the beginning of the semester following pair-programming guidelines (Campe et al., 2019). • Following the use-modify-create process, they first review a working example. Students first decompose the working model to see how if-statements work and then modify the code to fit into their own game. This step, although small, gives students a sense of accomplishment, satisfaction, and helps build their confidence for later stages. • The teacher floats around the room to help students as they test and debug their games. Pairs that finish early can either make further adjustments to their game or provide support to peers.
The lesson is wrapped up by introducing the next topic by asking a leading question: what if we have two rules instead of one (i.e., else-if)? 5 minutes	• Students receive the question for the next day, and this builds curiosity (i.e., inquiry arousal) for the next day's work.

Objectives

- Students will be able to describe the role of if-then statements in coding
- Students will be able to give an example of an if-then statement
- Students will be able to incorporate one if-then statement into their games
- Students will be able to describe the role of if-then statements in games as elements of games

Standards

- Computer Science Teachers Association (CSTA): (a) 1B-AP-10: Create programs that include sequences, events, loops, and conditionals. (b) 1B-AP-12: Modify, remix, or incorporate portions of an existing program into one's work, to develop something new or add more advanced features.
- GA-K-8 CS Standards (also found in Georgia Middle Grades Foundations of Interactive Design (MS-CS-FID) Course Number: 11.01300): (a) CSS.KC.6–8.20 Design, develop, debug, and implement computer programs. (b) CSS.KC.6–8.27: Create a functional game, using a game development platform, based on the storyboards, wireframes, and comprehensive layout. (c) CSS.CT.6–8.37 Use and compare simple coding control structures (e.g., if-then, loops).

Summary of Key Concepts on the Curriculum and the Lessons

We believe the following key concepts that we benefited from while designing and implementing our curriculum can inform future work that targets creating technology-rich learning experiences, including computer science education:

- The curriculum should be built on bottom-up: start with the basic concepts and move steadily up in complexity
- Building units around products is important. For example, in our curriculum, we had two units that led to a finished game each time.
- Create opportunities to build student interest and maintain engagement (i.e., following the ARCS model)
- Have specific goals for each lesson but allow for expandability. In other words, the pacing should allow for everyone to complete the lesson tasks within the lesson time, but if there are extra time, there should be opportunities for extra challenges
- Build for repeating key concepts throughout the curriculum.

Lessons Learned and Going Forward

Following our initial design and piloting and implementation in this section, we discuss what the design and the implementation phases taught us that are generalizable across other similar efforts, as well specific to this curriculum.

In this project our goal was to introduce students to real coding and real game-design through an industry standard software. Based on the feedback from our teachers, the implementation was more successful with 7th graders than 6th graders. Although we do not have the specific reasons at this point (will be discussed in future papers), we suspect that the level of complexity was cognitively beyond the capabilities of the younger age group. It should be noted, however, that through instructional approaches to coding such as pair programming some of the challenges can be addressed. For instance, we found that students working in dyads supporting each other cognitively and emotionally was essential for the success of the curriculum. We believe that the flow of the curriculum, as well as the theoretically sound design allowed for a successful implementation of challenging content through industry-standard software. This risk is highly associated with a big reward in that students through this experience can develop a more realistic identity as a computer programmer and game-designer and can in fact idealize themselves as future game-designers.

Another key aspect of this project was teacher professional development. Teaching the curriculum (implementation) requires teachers to develop some specific skills and knowledge, such as being proficient in Unity. Therefore, we believe that effective PD opportunities are necessary. As noted in our study (Hodges et al., 2022), the teachers in this project benefitted from weekly meetings with an expert and support from the researchers as they were teaching the curriculum. With an effective PD, it is highly likely that the outcomes of the curriculum would become more robust and transformative of student learning and motivation toward CS and coding. Another key concept in our project design was the perfect match between teachers' and students' learning experiences: teachers followed the same lesson plans and the curriculum as their students the previous year. This allowed teachers to experience the difficulties their students may face, develop strategies for how to overcome them, and be prepared to be flexible during their teaching.

Finally, we should note the difficulties related to hardware and software that is associated with such an innovative approach. Increasingly more school systems are switching to bring your own device (BYOD) or other approaches that tend to prefer tablet-like (e.g., Chromebooks) over full-fledged PCs. Although there are many benefits to this approach, in cases like ours where the innovation comes through proprietary software, these computers are not powerful enough. In fact, our implementation was delayed in some schools where they only had Chromebooks or other tablets. In addition, installation of unusual software is challenging some school systems. Therefore, steps should be taken ahead of the time to include the district IT personnel in the implementation process.

Conclusions

We believe the curriculum described in this paper can serve as a blueprint for other curricula for different software or higher order skills. Especially for teaching CS in middle schools, we believe a direct approach like ours can be beneficial in getting students more motivated to learn to continue CS due to its direct link to the game-design industry. We believe that the core concepts such as simple-to-difficult, and specific attention to motivational design (e.g., ARCS) can help overcome the difficulties brought about by the complexity of the content (and hardware). Finally, teacher learning is an important first step, and they should be given enough time to master the content (and the software) beforehand. We believe approaches to teacher PD where the teachers follow the same material as their learners can be good models for teacher PD in technology-rich topics. Our curriculum is publicly available and customizable. Similarly, our PD plans and documents are available as prototypes and can be accessed through our project website (https://projectgame.org).

Funding This study was funded by National Science Foundation (Grant #2027948). The authors declare they have no financial interests. All authors certify that they have no affiliations with or involvement in any organization or entity with any financial interest or non-financial interest in the subject matter or materials discussed in this manuscript.

References

Akcaoglu, M. (2014). Learning problem-solving through making games at the game design and learning summer program. *Educational Technology Research and Development, 62*(5), 583–600.

Akcaoglu, M. (2016). Design and implementation of the game-design and learning program. *TechTrends, 60*(2), 114–123.

Akcaoglu, M., & Green, L. S. (2019). Teaching systems thinking through game design. *Educational Technology Research and Development, 67*(1), 1–19.

Akcaoglu, M., & Kale, U. (2016). Teaching to teach (with) game design: Game design and learning workshops for preservice teachers. *Contemporary Issues in Technology and Teacher Education, 16*(1), 60–81.

Akcaoglu, M., & Koehler, M. J. (2014). Cognitive outcomes from the Game-Design and Learning (GDL) after-school program. *Computers & Education, 75*, 72–81.

Akcaoglu, M., Sonnleitner, P., Hodges, C., & Gutierrez, A. (2016). Teaching Complex Problem Solving Through Digital Game Design. In R. Zheng & M. Gardner (Eds.), *Handbook of research on serious games for educational applications* (pp. 217–233). IGN Publishing.

Basawapatna, A. R., Koh, K. H., & Repenning, A. (2010). Using scalable game design to teach computer science from middle school to graduate school. In *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education* (pp. 224–228).

Baytak, A., & Land, S. M. (2011). An investigation of the artifacts and process of constructing computer games about environmental science in a fifth grade classroom. *Educational Technology*

Research and Development, 59(6), 765–782. https://doi.org/10.1007/s11423-010-9184-z

Bosman, L., Chelberg, K., & Winn, R. (2017). How does service learning increase and sustain interest in engineering education for underrepresented pre-engineering college students? *Journal of STEM Education, 18*(2), 5–9.

Campe, S., Green, E. & Denner, J. (2019). K-12 Pair Programming Toolkit. ETR, Scotts Valley, CA.

Comber, O., Motschnig, R., Mayer, H., & Haselberger, D. (2019). Engaging students in computer science education through game development with Unity. In *2019 IEEE Global Engineering Education Conference (EDUCON)* (pp. 199–205). IEEE.

Deals, T. (2016). This engine is dominating the gaming industry right now. *The Next Web.* https://thenextweb.com/gaming/2016/03/24/engine-dominating-gaming-industry-right-now/. 12 Sept 2022.

Denner, J., Werner, L., & Ortiz, E. (2012). Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts? *Computers & Education, 58*(1), 240–249.

Denner, J., Campe, S., & Werner, L. (2019). Does computer game design and programming benefit children? A meta-synthesis of research. *ACM Transactions on Computing Education (TOCE), 19*(3), 19.

Dickson, P. E. (2015). Using unity to teach game development: When you've never written a game. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 75–80).

Earp, J. (2015). Game making for learning: A systematic review of the research literature. In *Proceedings of 8th international conference of education, research and innovation (ICERI2015)* (pp. 6426–6435).

Esquenazi, N. (2020). Coding Is collaborative and STEM education should be, too. *Bloomberg*. https://www.bloomberg.com/opinion/articles/2020-01-07/coding-is-collaborative-and-stem-education-should-be-too. 12 Sept 2022.

Goode, J., Margolis, J., & Chapman, G. (2014). Curriculum is not enough: The educational theory and research foundation of the exploring computer science professional development model. In *Proceedings of the 45th ACM technical symposium on Computer science education* (pp. 493–498).

Grover, S., & Basu, S. (2017). Measuring student learning in introductory block-based programming: Examining misconceptions of loops, variables, and boolean logic. In *Proceedings of the 2017 ACM SIGCSE technical symposium on computer science education* (pp. 267–272).

Guzdial, M. (2015). *Learner-centered design of computing education: Research on computing for everyone*. Morgan & Claypool.

Hartl, A. C., DeLay, D., Laursen, B., Denner, J., Werner, L., Campe, S., & Ortiz, E. (2015). Dyadic instruction for middle school students: Liking promotes learning. *Learning and Individual Differences, 44*, 33–39.

Hodges, Akcaoglu M., Allen, A., & Dogan, S. (2022). Teacher Self-efficacy During Professional Development for Game Design and Unity. *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education* V. 2, 1–1144. https://doi.org/10.1145/3478432.3499039.

Kafai, Y. B., & Burke, Q. (2016). *Connected gaming: What making video games can teach us about learning and literacy*. MIT Press.

Kirschner, P. A., & Neelen, M. (2020). Learning through play is more than play. 3-*Star Learning Experiences: An Evidence-Informed Blog for Learning Professionals.* https://3starlearningexperiences.wordpress.com/2020/02/18/learningthrough-play-is-more-than-play/. 12 Sept 2022.

Li, K., & Keller, J. M. (2018). Use of the ARCS model in education: A literature review. *Computers & Education, 122*, 54–62.

Mayer, R. E., & Wittrock, M. C. (2006). Problem solving. In P. A. Alexander & P. H. Winne (Eds.), *Handbook of educational psychology* (pp. 287–303). Lawrence Erlbaum Associates.

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2011). Habits of programming in scratch. In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education* (pp. 168–172).

Menekse, M. (2015). Computer science teacher professional development in the United States: A review of studies published between 2004 and 2014. *Computer Science Education, 25*(4), 325–350.

Penuel, W. R., Roschelle, J., & Shechtman, N. (2007). Designing formative assessment software with teachers: An analysis of the co-design process. *Research and Practice in Technology Enhanced Learning, 2*(01), 51–74.

Repenning, A. (2017). Moving beyond syntax: Lessons from 20 years of blocks programing in AgentSheets. *Journal of Visual Languages and Sentient Systems, 3*(1), 68–89.

Repenning, A., Webb, D., & Ioannidou, A. (2010). Scalable game design and the development of a checklist for getting computational thinking into public schools. In *Proceedings of the 41st ACM technical symposium on Computer science education* (pp. 265–269).

Repenning, A., Webb, D. C., Koh, K. H., Nickerson, H., Miller, S. B., Brand, C., ... & Gutierrez, K. (2015). Scalable game design: A strategy to bring systemic computer science education to schools through game design and simulation creation. *ACM Transactions on Computing Education (TOCE)*, *15*(2), 11.

Sullivan, F. R., & Denner, J. (2017). Why don't we do a better job of teaching computer science? *Education Week, 36*(36), 24.

Sweller, J. (2020). Cognitive load theory and educational technology. *Educational Technology Research and Development, 68*(1), 1–16. https://doi.org/10.1007/s11423-019-09701-3

Tlili, A., Essalmi, F., & Jemni, M. (2017). Towards applying Keller's ARCS model and learning by doing strategy in classroom courses. In E. Popescu, Kinshuk, M.K. Khribi, R. Huang, M. Jemni, N. Chen, D. G. Sampson (Eds) Innovations in Smart Learning (pp. 189–198). Springer. https://doi.org/10.1007/978-981-10-2419-1_26

Unity 3D (n.d.). Powering the real-time revolution. *Public Relations.* https://unity3d.com/public-relations. 12 Sept 2022.

Werner, L., Denner, J., Campe, S., & Torres, D. M. (2020). Computational sophistication of games programmed by children: A model for its measurement. *ACM Transactions on Computing Education (TOCE), 20*(2), 1–23.